John von Neumann Institute for Computing

Distributed Shared Memory in a Grid Environment

J.P. Ryan, B.A. Coghlan

published in

Parallel Computing:
Current & Future Issues of High-End Computing,
Proceedings of the International Conference ParCo 2005,
G.R. Joubert, W.E. Nagel, F.J. Peters, O. Plata, P. Tirado, E. Zapata (Editors),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **33**, ISBN 3-00-017352-8, pp. 129-136, 2006.

http://www.fz-juelich.de/nic-series/volume33

Distributed Shared Memory in a Grid Environment

John P Ryan [a], Brian A Coghlan[a],
{john.p.ryan, coghlan}@cs.tcd.ie

[a]Computer Architecture Group, Dept. of Computer Science, Trinity College Dublin, Dublin 2, Ireland.

1. Abstract

Software distributed shared memory (DSM) aims to provide the illusion of a shared memory environment when physical resources do not allow for it. Here we will apply this execution model to the Grid. Typically a DSM runtime incurs substantial overheads that result in severe degradation in performance of an application with respect to a more efficient message passing implementation. We examine mechanisms that have the potential to increase DSM performance by minimizing high-latency inter-process messages and data transfers. Relaxed consistency models are investigated, as well as the use of a grid information system to ascertain topology information. The latter allows for hierarchy-aware management of shared data and synchronization variables. The process of incremental hybridization, where more efficient message-passing mechanisms can incrementally replace those DSM actions that adversely effect performance, is also explored.

2. Introduction

The message passing programming paradigm enables the construction of parallel applications while minimizing the impact of distributing an algorithm across multiple processes by providing simple mechanisms to transfer shared application data between the processes. However, considerable burden is placed on the programmer whereby send/receive message pairs must be explicitly declared, and this can often be a source of errors. Implementations of message passing paradigms currently exist for grid platforms [11].

The shared memory paradigm is a simpler paradigm for constructing parallel applications, as it offers uniform access methods to memory for all user threads of execution, removing the responsibility of explicitly instrumenting the code with data transfer routines, and hence offers a less burdensome method to construct the applications. Its primary disadvantage is its limited scalability; nonetheless, a great deal of parallel software has been written in this manner. A secondary disadvantage has previously been the lack of a common programming interface, but this has been mitigated by the introduction of emerging standards in this area, such as OpenMP [1] . With OpenMP, compiler directives are used to parallelize serial code by explicitly identifying the areas of code that can be executed concurrently. That this parallelization can be done in an incremental fashion is an important feature in promoting the adoption of this standard.

These are the two predominant models for parallel computing. There are implementations of both paradigms for different architectures and platforms. Shared memory programming has the easier programming semantics, while message passing is more efficient and scalable (communication is explicitly defined and overheads such as control messages can be reduced dramatically or even eliminated). Previous work examined approaches to combine the message passing and shared-memory paradigms in order to leverage the benefits of both approaches, especially when the applications are executed in an environment such as a cluster of SMPs [19].

In contrast, Distributed Shared Memory (DSM) implementations aim to provide an abstraction of shared memory to parallel applications executing on ensembles of physically distributed machines. The application developer therefore leverages the benefits of developing in a style similar to shared memory, while harnessing the price/performance benefits associated with distributed memory platforms. Throughout the 1990's there were numerous research projects in the area of Software-only Distributed Shared Memory (S-DSM), e.g. Midway [6], Treadmarks [13], and Brazos [19]. However, little success has been achieved due to poor scalability and the lack of a common Application Programming Interface (API) [7].
The premise of grid computing is that distributed sites make available resources for use by remote users. A grid job may be run on one or more sites, and each site may consist of a heterogeneous group of machines. As Grids are composed of geographically distributed memory machines, traditional shared memory programs may not execute on multiple processors across multiple grid sites. DSM offers a potential solution, but numerous barriers exist to an efficient implementation on the Grid. However, if the paradigm could be made available, then according to [9], grid programming would be reduced to optimizing the assignment and use of threads and the communication system.
Our aim has been to explore a composition of the environments, with an OpenMP-compatible DSM system layered on top of a version of the now ubiquitous Message Passing Interface (MPI) standard, that can execute in a Grid environment. This choice reflects a desire for a tight integration with the message passing library and an awareness of the extensive optimization of MPI communications by manufacturers. Such a system would need to be multi-threaded to allow the overlap of computation and communication to mask the high-latencies, and to exploit the emergence of low-cost hardware such as multi-processor machines/multi-core processors.
Some of the main concerns have been to minimize the use of the high-latency communication channels between participating grid sites, to favour a lower message count with higher message payload, and also to avoid any need for specialised hardware support from the interconnect for remote memory access.

3. Grid DSM System Requirements

The design of a Grid S-DSM system must balance many factors, some conflicting. Some of these choices include the specification of the development interface, shared data management routines, the maintenance of shared data consistency across distributed processes, and the method of communication between DSM processes. With grid computing additional factors are introduced, e.g. grids may be composed of multiple architectures and platforms with different native data representations and formats.

3.1. Parallel Grid Applications and DSM

A DSM system must present the programmer with an easy-to-use and intuitive API so that the burden associated with the construction of a parallel application is minimized. This infers that the semantics should be as close to that of shared memory programming as possible, and that it should borrow from the successes and learn from the mistakes of previous DSM implementations.
For a DSM to gain acceptance, compliance with open standards is also necessary. Rather than having a direct implementation of one it may be more beneficial if the DSM forms the basis for a target of a parallelizing compiler that supports a programming standard, such as OpenMP. There are other projects have adopted this approach of source to source compilation [14]. Initial design requirements of the DSM can be identified by examining some OpenMP directives. The OpenMP code snippet below is an implementation the multiplication in parallel of two matrices.

```
int c[ROWSA][COLSA]
...
/*** Begin parallel section ***/
#pragma omp for
for (i = 0; i < ROWSA; i++){
  for(j = 0; j < COLSB; j++){
    c[i][j] = 0;
    for (k = 0; k < COLSA; k++)
      c[i][j] = c[i][j] +
                a[i][k] * b[k][j];
    }
  }
} /*** End parallel section ***/
```

Barrier primitives are used implicitly in OpenMP. Parallel application threads will wait at the end of the structured code block until all threads have arrived, except in some circumstances such as where a *nowait* clause has been declared. In order for concurrency to be allowed inside the parallel section, the shared memory regions must be writable by multiple writers, providing the accesses are non-competing. The code example above demonstrates the importance of allowing multiple writers to access shared data areas concurrently.
Additionally, mutual exclusion is required where parallel writes are possibly competing. In OpenMP, mutual exclusion areas are defined with the *master / single / critical* directives. A distributed mutual exclusion device (shared lock) is required to implement the *critical* directive, while the *master* and *single* OpenMP directives are functionally equivalent, and can be implemented using a simple if-then-else structure. Again implicit barriers are present at the end of these directives unless otherwise specified.

3.2. DSM Performance

The primary desire is the minimization of communication between nodes when maintaining consistency of shared data between DSM processes. Thus the selection of the consistency model is a prime determinant in the overall performance of the DSM system. This requires a relaxed consistency model to be employed, where data and synchronization operations are clearly distinguished, and data is only made consistent at synchronization points. The most notable are Release Consistency (RC) [10] Lazy-Release (LRC) [13], and Entry Consistency (EC) [6] models. The choice of which to use generally involves a trade-off between the complexity of the programming semantics, and the volume of overall control and consistency messages generated by the DSM.
Entry consistency is similar to lazy-release consistency in its use of synchronization primitives to direct coherency operations to shared data, but adopts an even more relaxed approach, whereby shared data is explicitly associated with a synchronization primitive(s) and is made consistent when such operations are performed; hence, minimal coherency messages are generated. Studies show that EC and LRC can on average produce the same performance on clusters of computers of a modest size [3]. It is important to note that the application's access pattern to shared memory is a determining factor. Entry consistency introduces additional programming complexity, but [16] examines the same application with EC and LRC, which clearly demonstrates the better performance of EC with respect to volume of messages generated. This is an important consideration if an application is executing across multiple sites with high-latency communication links.
Multiple-writer protocols attempt to increase concurrency by allowing multiple writes to locations

residing on the same coherence unit, so addressing the classical DSM problem of false sharing. Multiple-writer entry consistency attempts to combine the benefits of both models while addressing associated problems that have been identified for the LRC and EC protocols [4]. It has been demonstrated that significant performance gains can be achieved by employing this technique [18]. This protocol must be provided by the DSM in order to achieve true concurrency.

3.3. Use of a Grid Information & Monitoring System

There are tools available for the monitoring of message passing applications, such as MPICH's Jumpshot, but these are unsuitable for execution across geographically dispersed sites. It has been shown that when MDS, the grid information component of the Globus Toolkit, was integrated with a MPI implementation, dramatic improvements were obtained in the performance of a number of MPI collective operations through the use of hierarchy awareness [12]. If a Grid DSM can make use of services in this way then it should prove beneficial. Some of the benefits of integrated environmental awareness include the efficient implementation of global barrier synchronization, load-balancing using per-site lock management optimisation, effective per-site caching & write collection of shared data, and communication-efficient consistency updates.
The Relational Grid Monitoring Architecture (R-GMA) [17], is a relational grid information system, where information can be published via numerous distributed producers and accessed by a single consumer. R-GMA has successfully been used to enable the monitoring of MPI applications in a grid environment using GRM/PROVE [15]. Runtime monitoring and hierarchy awareness for DSM applications could be provided in a similar manner.

4. Implementation

Our prototype Software-DSM system is called SMG (Shared Memory for Grids). The MPI message passing library is currently utilized for system communication as it provides a stable interface, and support exists for a execution of MPI applications in a grid environment. The SMG API implementation draws on previous DSM APIs and consists of initialization & finalization function (essentially wrappers for the underlying communication routines), shared memory allocation functions, and synchronization operations. To maximise portability, the DSM implementation only uses standard libraries such as the POSIX thread, MPI, and the standard C libraries. No special compiler or kernel modifications are required for the DSM implementation itself. The development platform is the Linux operating system.
The code example below shows how the matrix multiply algorithm that was implemented in the previous section using OpenMP directives can now be implemented using the SMG library. SMG barriers appear before and after the parallel section. The resultant matrix *c* is explicitly declared and bound to the use of the synchronization variable *SMG_BARRIER_01*. When this barrier is reached the modifications to the shared variables are synchronized, checked for conflicts, and propagated among the nodes.

```
SMG_malloc(SMG_OBJ_001,sizeof(int),(COLSA * COLSB),&c_ptr,
            (ENTRY | BARRIER_NAMED), SMG_BARRIER_01);
// c initialised to c_ptr
...
SMG_barrier_(SMG_BARRIER_01);
for (i = start; i < end; i++)
  for(j = 0; j < COLSB; j++){
```

```
      c[i][j] = 0;
      for (k = 0; k < COLSA; k++)
        c[i][j] = c[i][j] +
                  a[i][k] * b[k][j];
    }
SMG_barrier_(SMG_BARRIER_01);
```

Although an OpenMP compiler targeting the SMG system has yet to be implemented it is envisaged that it would produce valid SMG code similar to the example above. Usage of SMG locks is not demonstrated here, but the single-reader/multiple-writer protocol is supported. Upon exclusive access of a lock by a process, writes can occur to the shared data that may be bound to that particular lock. Upon release of the lock and its subsequent acquire by another process all modifications made in the previous duration will be propagated to the requesting process.

4.1. Consistency and Coherency

EC increases the programming burden when compared with other relaxed consistency models such as LRC. However, its semantics can lend itself well to targeting by an OpenMP compiler, and for this reason as well as its lower message volumes it is the consistency protocol of choice. Write-update coherency is employed by default when an EC protocol is used. Multi-writer EC is supported when the shared data is associated with a barrier synchronization primitive, while lock primitives support a single-writer.

In S-DSMs, write-trapping is the process of detecting modifications made to shared data, while write-collection is the process by which the updates required to maintain consistency at remote processes are generated. The implementation of these concepts follows a similar approach to that adopted in Treadmarks [13] with some small modifications. Write-trapping is achieved by setting the protection level of the shared object, where the minimum granularity is at the virtual page level up to the total size of the object if it occupies more than one page. Upon the first write to a variable in a shared region a twin, or part thereof, is generated.

When the synchronization object that the shared memory object is bound to performs a release a diff is generated, by comparing the twin and the current 'dirty' state. This is used to minimize the message traffic for coherence updates. If topology information is available then hierarchy awareness can be applied in barrier operations, i.e. as arrival notifications are received the diffs can be merged at intermediate nodes, thus reducing the processing bottleneck at the root node level. Otherwise, a traditional tree-structured barrier is employed.

Multi-writer entry consistency is supported for barrier primitives as demonstrated in the code segment above; this is equivalent to OpenMP parallel *for* constructs. Basic user-specified alteration to write trapping and write collection methods are allowed, and when more integration with the information system occurs, this user control will enable application-level optimizations where access patterns to shared data are irregular [4].

4.2. Communication

Communication between processes is via an implementation of the SMG_comm interface. Thus far only a MPI version of this has been developed. Exploring the use of MPICH for the communication between DSM processes executing on distributed nodes allows for the exploitation of an optimized and stable message passing library, and also leveraging of useful MPI resources such as profiling tools and debuggers; its use also insulates the system from some platform dependencies and will ease porting to other architectures and platforms in the future.

Unfortunately the current Grid enabled version of MPICH, MPICH-G2, is based on the MPICH distribution (currently version 1.2.7), which has no support for multi-threaded MPI applications. This makes optimisations such as hybridizing (see 4.4) near impossible, as the DSM system thread requires the MPI communication channel, and so can only be used in MPI_THREAD_FUNNELLED mode. Implementations exist that provide a thread-safe MPI implementation, however they are not grid-enabled. Other MPI implementations are soon expected to support multi-threading.

4.3. Environment Awareness

An information system is required if the DSM is to be hierarchy-aware, otherwise the topology assumes the flat model typical of an MPI application. Both MDS and R-GMA adhere to the GLUE schema [2]. Either (or both) are a good basis for hierarchy awareness. The SMG DSM prototype initially uses R-GMA, principally for its support for relational queries and dynamic schemas. It uses the GLUE schema to obtain environmental information from GLUE compliant R-GMA information producers. To enable monitoring of the user applications, SMG schemas are defined so that monitoring information can be published using the R-GMA producer API, and viewed using the standard R-GMA browser. Nonetheless the information system is hidden by wrappers, and other systems, such as MDS, can be used by implementing the required wrapper APIs.
The SMG system makes use of R-GMA to produce and consume data in a similar manner. Topology information is consumed at startup, while DSM system and application information can be produced using the defined APIs. A tool has been developed to access application logging information allowing for runtime analysis. A further aim has been to use the information system to create topology/hierarchy awareness and runtime support of applications, where the developer wishes topology information to be exposed to the user application code to allow for optimizations at that level.

4.4. Incremental Hybridization

Using the information system allows for the runtime profiling of user applications, and allows for other tools such as deadlock detectors to be developed. We have constructed a user interface tool that can identify locations in an application where data access behaviour results in severe performance penalties. As we can monitor the variables and the code areas where these are used we can direct a *hybridization* process whereby the user replaces shared memory code with message passing, resulting in performance gains. This can be done in a localized fashion, and possibly in the future in conjunction with parallelization with OpenMP.
The hybridization GUI works by directing the user to the locations where the majority of communication occurs (so identifying the participating processes), and also to the shared variables that are responsible for the communication. The code browser illustrates the 'hot-spots' in the application and the the objects responsible. This allows for incremental and localized optimisations.
The developer can examine the interval between synchronization operations, such as between barriers, and view the volume of data transfers generated between processes. From other information gathered during the interval they can identify the memory objects causing the communication.If message passing and shared memory paradigms are used at different times during the application's execution for the same variables, then the shared regions must be made consistent after message passing is used.
Another feature is the highlighting of fragmented shared memory use which may be the result of poor algorithm design/implementation. This can be identified when a shared memory region is repeatedly modified by multiple processes in a fragmented fashion. Such a scenario may occur when a developer, unaware of the potential differences between C and Fortran languages (row and column ordering) transposes a matrix multiplication algorithm.

5. Performance

There is no grid-enabled MPI implementation that currently supports multi-threaded applications, we therefore utilize a non-grid flavour of MPI. The SMG DSM is currently being tested in a virtual grid environment, with the number of sites configurable at runtime. System performance measurements are not representative, since DSM communication is unable to avail of blocking receive MPI calls. However, the significant memory overhead associated with the DSM (update catalogue) persists, as will the substantial processing & memory requirements for consistency actions (twinning/diffing). This can be overcome at the expense of increased coherence message volumes.
Table 1 below shows the total message count resulting from the SMG implementation of a simple Laplace example compared with that of a similar message passing implementation, each involving the same number of iterations. The message volumes compare favourably with the difference being extra messaging incurred by the DSM at initialisation & finalization. Additionally, the usefulness of some of the strategies we employ (such as hierarchy-awareness) can be demonstrated; where the simulation involves *N* sites, the number of inter-site messages per barrier is maintained at *2N*.

Number Processes	4	8	16	32
MPI	603	1407	3015	6231
SMG	612	1428	3060	6343

Table 1
Messaging costs for parallel Laplace

6. Conclusions and Summary

Grids are starting to impact on mainstream parallel computing. If this trend is set to continue then improved tools and development environments must be implemented. We believe that there will be numerous approaches to constructing grid applications, be it message passing, shared memory, or DSM. Clearly none of these in isolation will provide the perfect fit, but rather an ensemble.
DSM implementations such as Treadmarks, Munin, and Midway were written for compute clusters, not for Grid computing. Efforts at making hierarchy-aware DSM consistency protocols do exist [5], as well as efforts to allow OpenMP to run on distributed memory machines [14]. There have been other attempts at providing a shared memory model for wide area computing [8], and efforts are underway to implement the MPI-2 standard, which includes specifications for remote memory access and one sided communications.
In the SMG system, we are attempting to demonstrate the potential advantages when message-passing and DSM programming paradigms are combined in the grid environment. The goal is to reduce the programming burden, and allow it to be followed by incremental optimisation. If this is achieved, it will promote the use of grids by allowing the exploitation of the very large collection of existing shared memory codes, and allow for easier parallelization/grid-enabling of serial codes through the use of the OpenMP standard. Coherence overheads are comparable with other DSM implementations but this will be improved when environmental awareness techniques and alterable write trapping/collection techniques are further improved.
Future support for heterogeneity is vital in order to further the cause. This problem is non-trivial, as shared data would need to be strongly typed. We believe that this is one area where compiler support would prove beneficial.

References

[1] Homepage of OpenMP initiative. http://www.openmp.org.

[2] The GLUE Schema. http://www.cnaf.infn.it/sergio/datatag/glue/index.htm.

[3] S. Adve, A.L. Cox, S. Dwarkadas, R. Rajamony, and W. Zwaenepoel. A Comparison of Entry Consistency and Lazy Release Consistency Implementations. In *Proceedings of the Second High Performance Computer Architecture Conference*, pages 26–37, Feb 1996.

[4] Christina Amza, A.L. Cox, Sandhya Dwarkadas, Li-Jie Jin, Karthick Rajamani, and Willy Zwaenepoel. Adaptive Protocols for Software Distributed Shared Memory. *Journal of the IEEE, Special Issue on Distributed Shared Memory*, pages 467–475, Mar 1999.

[5] Gabriel Antoniu, Luc Boug, and Sbastien Lacour. Making a DSM Consistency Protocol Hierarchy-Aware: an Efficient Synchronization Scheme. In *Proc. Workshop on Distributed Shared Memory on Clusters (DSM'03)*, pages 516–523, May 2003.

[6] B.N. Bershad, M.J. Zekauskas, and W.A. Sawdon. The midway distributed shared memory system. In *Procs. of COMPCON. The 38th Annual IEEE Computer Society International Computer Conference*, pages 528–537, Feb 1993.

[7] J. B. Carter, D. Khandekar, and L. Kamb. Distributed Shared Memory: Where We Are and Where We Should Be Headed? In *Fifth Workshop on Hot Topics in Operating Systems*, pages 119–122, 1995.

[8] DeQing Chen, Chunqiang Tang, Xiangchuan Chen, Sandhya Dwarkadas, and Michael L. Scott. Multi-level Shared State for Distributed Systems. In *Procs. of 31st Int. Conference on Parallel Processing (ICPP'02)*, August 2002.

[9] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan-Kaufmann, 1999.

[10] K. Gharachorloo, D. Lenoski, J. Laudon, P.B. Gibbons, A. Gupta, and J.L. Hennessy. Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors. In *25 Years ISCA: Retrospectives and Reprints*, pages 376–387, 1998.

[11] N. Karonis, B. Toohen, and I. Foster. MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface. *Journal of Parallel and Distributed Computing*, 63(5):551–563, May 2003.

[12] N. T. Karonis, B. R. de Supinski, I. Foster, W. Gropp, E. Lusk, and J. Bresnahan. Exploiting Hierarchy in Parallel Computer Networks to Optimize Collective Operation Performance. In *Proc. of the 14th Int'l Conference on Parallel and Distributed Processing Symposium (IPDPS-00)*, pages 377–386, 2000.

[13] P. Keleher, S. Dwarkadas, A. L. Cox, and W. Zwaenepoel. TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems. In *Proc. of the Winter 1994 USENIX Conference*, pages 115–131, 1994.

[14] H. Lu, Y. C. Hu, and W. Zwaenepoel. OpenMP on network of workstations. In *Proc. of Supercomputing'98*, 1998.

[15] N. Podhorszki and P. Kacsuk. Monitoring Message Passing Applications in the Grid with GRM and R-GMA. *Recent Advances in Parallel Virtual Machine and Message Passing Interface (PVM/MPI)*, pages 603–610, 2003.

[16] Jelica Protic and Veljko Milutinovic. Entry consistency versus lazy release consistency in dsm systems: Analytical comparison and a new hybrid solution. In *IEEE Workshop on Future Trends of Distributed Computing Systems*, pages 78–83, Oct 1997.

[17] S. Fisher et al. R-GMA: A Relational Grid Information and Monitoring System. Technical Report WP3-2003-01-14, 2nd Cracow Grid Workshop, DATAGRID, Jan 2003.

[18] H. S. Sandhu, T. Brecht, and D. Moscoso. Multiple Writers Entry Consistency. In *Proc. of the Int'l Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA'98)*, volume I, pages 355–362, 1998.

[19] E. Speight, H. Abdel-Shafi, and J.K. Bennett. An Integrated Shared-Memory/Message Passing API for Cluster-Based Multicomputing. In *Procs. of the Second International Conference on Parallel and Distributed Computing and Networks (PDCN)*, Brisbane, Australia, 1998.